```
#include <avr/io.h>
#include <avr/interrupt.h>

// get voltages from 0-255 pwm settings, timer0 count a.  using my very basic transistor circuit
//      start with 8 settings to see if this works at all
// remember to rewire and change programs for moving the connections to the LED columns
//     from PD5, PD6, PD7, PB0 to PC2, PC3, PC4, PC5.
//     to open up the OC0A and OC0B PWM pins.

// the periods frequencies of notes 440Hz base, repeated octaves, binary multiples of the number of cycles at 78,125Hz (PWM duty cycle).
// note 72 is base note for octave 0, note 0.  note 0 is MIDI note 15, note 112 is MIDI note 127.
// lookup table array - 122 entries (0-121), 440Hz reference note is note 54, middle C is note entry 45 (299 periods of 78,125 Hz)
// when I learn how, put this in Flash (PROGMEM method)
uint8_t SvNoteWaveLength78K[] = { 251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,
211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,
211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237,224,
211,199,188,178,168,158,149,141,133,251,237,224,211,199,188,178,168,158,149,141,133,251,237 } ;

// lookup table array - Sine Wave - 256 steps (0-255) with values from 0-255
// when I learn how, put this in Flash (PROGMEM method)
uint8_t SvSineWaveLookup256[] = { 127,131,134,137,140,143,146,149,152,155,158,162,165,167,170,173,176,179,182,185,188,190,193,196,198,201,203,
206,208,211,213,215,218,220,222,224,226,228,230,232,234,235,237,238,240,241,243,244,245,246,248,249,250,250,251,252,253,253,254,254,254,255,255,
255,255,255,255,255,254,254,254,253,253,252,251,250,250,249,248,246,245,244,243,241,240,238,237,235,234,232,230,228,226,224,222,220,218,215,213,
211,208,206,203,201,198,196,193,190,188,185,182,179,176,173,170,167,165,162,158,155,152,149,146,143,140,137,134,131,127,124,121,118,115,112,109,
106,103,100,97,93,90,88,85,82,79,76,73,70,67,65,62,59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,20,18,17,15,14,12,11,10,9,7,6,5,5,4,3,2
,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,25,27,29,31,33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70,73,76,
79,82,85,88,90,93,97,100,103,106,109,112,115,118,121,124 } ;

// two arrays, one that is being activley "played", the other that is being prepared for use
// initial settings here are the maximum size that is intended to be used
// SvArrayElements[] is the number of elements in the PlayArray[] array (initial value is 256, to match the initial PlayArray[] array, elements
are numbered 0-255)
// the value of SvArrayElements[] is taken from the lookup table array SvNoteWaveLength78K[]
volatile uint8_t SvPlayArray[2][256] = { { 127,131,134,137,140,143,146,149,152,155,158,162,165,167,170,173,176,179,182,185,188,190,193,196,198,
201,203,206,208,211,213,215,218,220,222,224,226,228,230,232,234,235,237,238,240,241,243,244,245,246,248,249,250,250,251,252,253,253,254,254,254,
255,255,255,255,255,255,255,254,254,254,253,253,252,251,250,250,249,248,246,245,244,243,241,240,238,237,235,234,232,230,228,226,224,222,220,218,
215,213,211,208,206,203,201,198,196,193,190,188,185,182,179,176,173,170,167,165,162,158,155,152,149,146,143,140,137,134,131,127,124,121,118,115,
112,109,106,103,100,97,93,90,88,85,82,79,76,73,70,67,65,62,59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,20,18,17,15,14,12,11,10,9,7,6,5
,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,25,27,29,31,33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70
,73,76,79,82,85,88,90,93,97,100,103,106,109,112,115,118,121,124 } ,  { 127,131,134,137,140,143,146,149,152,155,158,162,165,167,170,173,176,179,
182,185,188,190,193,196,198,201,203,206,208,211,213,215,218,220,222,224,226,228,230,232,234,235,237,238,240,241,243,244,245,246,248,249,250,250,
251,252,253,253,254,254,254,255,255,255,255,255,255,255,254,254,254,253,253,252,251,250,250,249,248,246,245,244,243,241,240,238,237,235,234,232,
230,228,226,224,222,220,218,215,213,211,208,206,203,201,198,196,193,190,188,185,182,179,176,173,170,167,165,162,158,155,152,149,146,143,140,137,
134,131,127,124,121,118,115,112,109,106,103,100,97,93,90,88,85,82,79,76,73,70,67,65,62,59,57,54,52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,20,
18,17,15,14,12,11,10,9,7,6,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,25,27,29,31,33,35,37,40,42,44,
47,49,52,54,57,59,62,65,67,70,73,76,79,82,85,88,90,93,97,100,103,106,109,112,115,118,121,124 } } ;
volatile uint8_t SvArrayElements[] = { 255, 255 };

// SvNoteHighNOTLow[] - for ISR, indicates which method of counting cycles to use.
// Low notes (notes 0-47) count through SvPlayArray[] by binary multiples(?) of 78K cycles.  (1 << SvNoteFactor) 78K cycles per array element.
// High notes (notes 48-121) move to the next element of SvPlayArray[] each 78K cycle.
volatile uint8_t SvNoteHighNOTLow[] = { 1, 1 };  // just a binary value
volatile uint8_t SvNoteFactor[2];  // initializes to 0

volatile uint8_t SvArrNoteDuration[] = { 63, 63 };  // numbered in SvDurationResolution units (the fractional note durations of BPM), counting
from 0 to total counts, up to 255
volatile uint16_t SvDurationResolution[] = { 305, 305 };  // number of 78K cycles for in the minimum fractional note duration

volatile uint8_t SvLiveArr;  // binary value, selects which SvPlayArray[] is currently being "played"
volatile uint8_t SvNextLiveArr;  // binary value, selects which SvPlayArray[] is next
volatile uint8_t SvArrayLiveCounter;  // counts which element is currently being "played", from 0 up to SvArrayElements[]
volatile uint8_t SvFactor78KCounter;  // for Low notes, counts the current count of 78K cycles, from 0 up to (1 << SvNoteFactor[])-1
volatile uint8_t SvLiveNoteDurationCounter;  // counts how long the current note has been played, when SvLiveNoteDurationCounter =
SvArrNoteDuration[], toggle SvLiveArr

volatile uint16_t SvLive78kCyclesCounter;  // counts how many 78K cycles have been played in the current duration resolution unit
volatile uint8_t SvStartCalculatingNextNote;  // flag that indicates that SvLiveArray has been toggled, and it's time to start calculating the
next note's SvPlayArray[]

struct NoteStruct {
```

```c
    uint8_t SvNoteNum;
    uint16_t SvNoteDuration;  // counting from 0 to total counts?
    uint8_t SvNoteVolume;
};

uint8_t SvNumberOfNotesToPlay;

struct NoteStruct SvNotesToPlay[4];
struct NoteStruct SvNoteForPlayArray;

void SfToneGenerator (struct NoteStruct SvNoteForPlayArray);
void Start_Timer0_PWM (void);

int main (void) {

    uint16_t SvMusicTempoBPMxSub;
    uint8_t SvNoteStructNum;

    //avr-libc assures us that variables will automatically be initiallized to zero

    SvStartCalculatingNextNote = 1;

    SvLiveArr = 0;  // initialized to 0 for clarity
    SvNextLiveArr = 1;

    // the smallest value for note duration should be no less than 1/305 of a second, 256 "78K cycles".
    // this minimum can be increased up to 512 "78K cycles", to work out various BPM settings.
    // beginning example 120 BPM with 128th notes.  1/256 of a second minimum note duration, this is 305 "78K cycles".
    SvMusicTempoBPMxSub = 120 * 128;  // beats per minute times number of fractional note durations, if using a faster than 120 BPM, use less than 128 fractional note durations
    SvDurationResolution[0] = ( (uint32_t)4687500 / SvMusicTempoBPMxSub );  // 4,687,500 = 78,125 * 60
    SvDurationResolution[1] = ( (uint32_t)4687500 / SvMusicTempoBPMxSub );  // 4,687,500 = 78,125 * 60
    //SvDurationResolution = 2441;  // temp test for math, 2441 is 32 per second ( 78,125 PWM cycles-per-second * 60 seconds ) / (120BPM * 16) for 16th notes resolution
    //SvDurationResolution = 305;  // temp test for math, 305 is 256 per second ( 78,125 PWM cycles-per-second * 60 seconds ) / (120 BPM * 128)
    for 128th notes resolution

    SvNumberOfNotesToPlay = 4;

    SvNotesToPlay[0].SvNoteNum = 45;
    SvNotesToPlay[0].SvNoteDuration = 64;
    SvNotesToPlay[0].SvNoteVolume = 127;

    SvNotesToPlay[1].SvNoteNum = 48;
    SvNotesToPlay[1].SvNoteDuration = 64;
    SvNotesToPlay[1].SvNoteVolume = 127;

    SvNotesToPlay[2].SvNoteNum = 51;
    SvNotesToPlay[2].SvNoteDuration = 64;
    SvNotesToPlay[2].SvNoteVolume = 127;

    SvNotesToPlay[3].SvNoteNum = 54;
    SvNotesToPlay[3].SvNoteDuration = 64;
    SvNotesToPlay[3].SvNoteVolume = 127;

    sei();  // DUH!
    Start_Timer0_PWM ();

    // test
    SvLiveArr = 0;
    SvNextLiveArr = 1;
    SfToneGenerator(SvNotesToPlay[0]);  // fill the SvPlayArray[0] with useful data
    SvLiveArr = 1;
    SvNextLiveArr = 0;
    SfToneGenerator(SvNotesToPlay[1]);  // fill the SvPlayArray[1] with useful data
    SvLiveArr = 0;
    SvNextLiveArr = 1;
    // end test

    while (1) {
```

```c
        if (SvStartCalculatingNextNote == 1) {
            SvArrayLiveCounter = 0;
            SvFactor78KCounter = 0;
            SvLive78kCyclesCounter = 0;
            SvLiveNoteDurationCounter = 0;
            SfToneGenerator(SvNotesToPlay[SvNoteStructNum]);
            if (SvNoteStructNum = SvNumberOfNotesToPlay) {
                SvNoteStructNum = 0;  // start over at beginning of list
            }
            else SvNoteStructNum++;
        }
    }
}

void SfToneGenerator (struct NoteStruct SvNoteForPlayArray) {  //uint8_t SvNoteNum, uint16_t SvNoteDuration, uint16_t SvNoteVolume

    uint8_t SvArrayCount;  // for loop counter
    uint8_t SvCyclesTotal;  // = ( 1 << SvNoteFactor[] )
    uint16_t temp1;
    uint8_t temp2;
    uint8_t temp3;

    SvStartCalculatingNextNote = 0;  // calculating the next note SvPlayArray is started
    SvArrNoteDuration[SvNextLiveArr] = SvNoteForPlayArray.SvNoteDuration;  // currently going to set up the NON live array

    if (SvNoteForPlayArray.SvNoteNum < 48) {

        SvNoteHighNOTLow[SvNextLiveArr] = 0;  // false, this is not a high note, it is a low note
        SvNoteFactor[SvNextLiveArr] = (60-SvNoteForPlayArray.SvNoteNum) / 12;  // (1 << SvNoteFactor[]) is the number of 78K steps per entry in
        SvPlayArray[] (count by (1 << SvNoteFactor[]) when playing back
        SvArrayElements[SvNextLiveArr] = SvNoteWaveLength78K[SvNoteForPlayArray.SvNoteNum];
        for (SvArrayCount = 0; SvArrayCount < SvArrayElements[SvNextLiveArr]; SvArrayCount++) {
            temp1 = (SvArrayCount << 8);
            SvPlayArray[SvNextLiveArr][SvArrayCount] = (SvSineWaveLookup256[ temp1 / SvArrayElements[SvNextLiveArr] ] * SvNoteForPlayArray.
            SvNoteVolume) >> 8;
        }
    }
    else {  // SvNoteNum > 47

        SvNoteHighNOTLow[SvNextLiveArr] = 1;  // true, this is a high note, not a low note
        SvNoteFactor[SvNextLiveArr] = (SvNoteForPlayArray.SvNoteNum - 36) / 12;  // (1 << SvNoteFactor) is number of Sine Wave Cycles in the
        SvPlayArray
        SvArrayElements[SvNextLiveArr] = SvNoteWaveLength78K[SvNoteForPlayArray.SvNoteNum];
        SvCyclesTotal = ( 1 << SvNoteFactor[SvNextLiveArr] );  // for use in calculation in the next line
        for (SvArrayCount = 0; SvArrayCount < SvArrayElements[SvNextLiveArr]; SvArrayCount++) {
            temp1 = (SvArrayCount << 8);
            temp2 = (temp1 / SvArrayElements[SvNextLiveArr]);
            temp3 = (((SvArrayCount * SvCyclesTotal) / SvArrayElements[SvNextLiveArr]) << (8 - SvNoteFactor[SvNextLiveArr]));
            SvPlayArray[SvNextLiveArr][SvArrayCount] = (SvSineWaveLookup256[ (temp2 - temp3) * SvCyclesTotal] * SvNoteForPlayArray.SvNoteVolume)
             >> 8;
            //SvPlayArray[SvNextLiveArr][SvArrayCount] = (SvSineWaveLookup256[ (((SvArrayCount << 8) / SvArrayElements[SvNextLiveArr]) -
            (((SvArrayCount * SvCyclesTotal) / SvArrayElements[SvNextLiveArr]) << (8 - SvNoteFactor[SvNextLiveArr]))) * SvCyclesTotal] *
            SvNoteForPlayArray.SvNoteVolume) >> 8;
        }
    }
}

void Start_Timer0_PWM (void) {

    DDRD = (1 << DDD6);  // set OC0A PWM pin as output (PD6, pin 12)
    TCCR0A |= (1 << WGM01) | (1 << WGM00);  // set timer 0 for Fast PWM mode
    TCCR0A |= (1 << COM0A1);  // Clear OC0A on Compare Match, set OC0A at BOTTOM
    TIMSK0 |= (1 << TOIE0);  // Enable PWM overflow interrupt, REMEMBER TO START ALL INTERRUPTS sei();
    OCR0A = 0;  // set Output Compare Register A to 0 (one cycle on per 256)
    //OCR0A = 0x00;  // test for designing and verifying the transistor amplifier circuit
    TCCR0B |= (1 << CS00);  // start PWM timer with no prescaler
}

ISR(TIMER0_OVF_vect) {
    if (SvNoteHighNOTLow[SvLiveArr]) {  // this is a High note, notes > 47.  count through an element in the SvPlayArray[] for every 78k cycle.

        OCR0A = SvPlayArray[SvLiveArr][SvArrayLiveCounter];
        if (SvArrayLiveCounter == SvArrayElements[SvLiveArr] ) {
            SvArrayLiveCounter = 0;
        }
        else SvArrayLiveCounter++;

        if (SvLive78kCyclesCounter == SvDurationResolution[SvLiveArr] ) {
            SvLive78kCyclesCounter = 0;
            if (SvLiveNoteDurationCounter == SvArrNoteDuration[SvLiveArr] ) {
                SvLiveNoteDurationCounter = 0;
                if (SvLiveArr == 1) {  // toggle to the other array, it's now the "live" array being "played"
                    SvLiveArr = 0;
                    SvNextLiveArr = 1;
                }
                else {
                    SvLiveArr = 1;
                    SvNextLiveArr = 0;
                }
                SvStartCalculatingNextNote = 1;  // flag indicates it's time to start calculating the new next note SvPlayArray
            }
            else SvLiveNoteDurationCounter++;
        }
        else SvLive78kCyclesCounter++;
    }
    else {  // this is a low note, notes < 48.  count through an element in the SvPlayArray[] for every (1 << SvNoteFactor) times 78k cycle.

        if (SvFactor78KCounter == (1 << SvNoteFactor[SvLiveArr] )-1 ) {
            SvFactor78KCounter = 0;
            OCR0A = SvPlayArray[SvLiveArr][SvArrayLiveCounter];
            if (SvArrayLiveCounter == SvArrayElements[SvLiveArr] ) {
                SvArrayLiveCounter = 0;
            }
            else SvArrayLiveCounter++;
        }
        else SvFactor78KCounter++;

        if (SvLive78kCyclesCounter == SvDurationResolution[SvLiveArr] ) {
            SvLive78kCyclesCounter = 0;
            if (SvLiveNoteDurationCounter == SvArrNoteDuration[SvLiveArr] ) {
                SvLiveNoteDurationCounter = 0;
                if (SvLiveArr == 1) {  // toggle to the other array, it's now the "live" array being "played"
                    SvLiveArr = 0;
                    SvNextLiveArr = 1;
                }
                else {
                    SvLiveArr = 1;
                    SvNextLiveArr = 0;
                }
                SvStartCalculatingNextNote = 1;  // flag indicates it's time to start calculating the new next note SvPlayArray
            }
            else SvLiveNoteDurationCounter++;
        }
        else SvLive78kCyclesCounter++;
    }
}

//FOR REFERENCE WHILE I BUILD THIS ISR
//volatile uint8_t SvArrNoteDuration[] = { 64, 64 };  // numbered in SvDurationResolution units (the fractional note durations of BPM)
//volatile uint16_t SvDurationResolution[] = { 305, 305 };  // number of 78K cycles for in the minimum fractional note duration

//volatile uint8_t SvLiveArr;  // binary value, selects which SvPlayArray[] is currently being "played"
//volatile uint8_t SvNextLiveArr;  // binary value, selects which SvPlayArray[] is next
//volatile uint8_t SvArrayLiveCounter;  // counts which element is currently being "played", from 0 up to SvArrayElements[]
//volatile uint8_t SvFactor78KCounter;  // for Low notes, counts the current count of 78K cycles, from 0 up to (1 << SvNoteFactor[])-1
//volatile uint8_t SvLiveNoteDurationCounter;  // counts how long the current note has been played, when SvLiveNoteDurationCounter =
SvArrNoteDuration[], toggle SvLiveArr
```

```c
//volatile uint8_t SvLive78kCyclesCounter;  // counts how many 78K cycles have been played in the current duration resolution unit
//volatile uint8_t SvStartCalculatingNextNote;  // flag that indicates that SvLiveArray has been toggled, and it's time to start calculating the next note's SvPlayArray[]
```